Unisenac
Centro Universitário RS

Senac Fecomércio Sesc

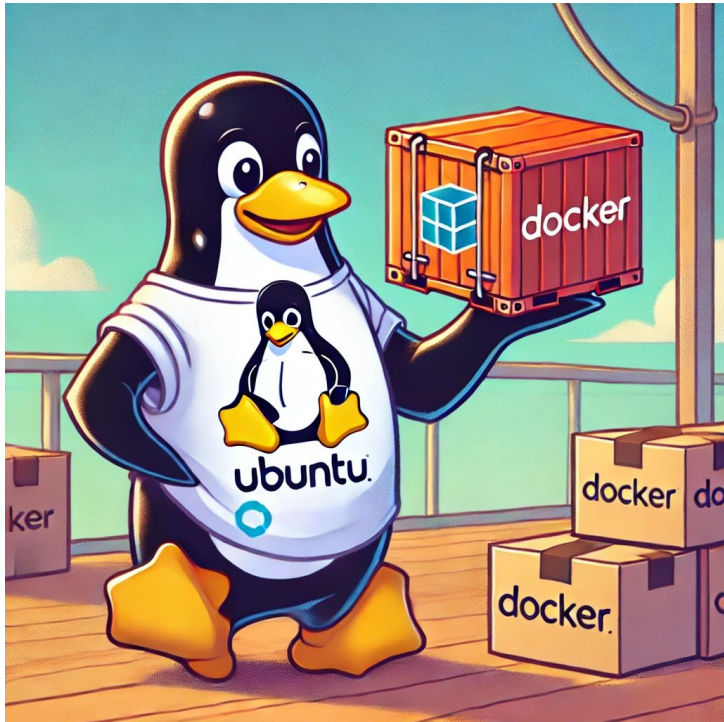
Computação em Nuvem

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

Prof. Guto Muniz

Introdução

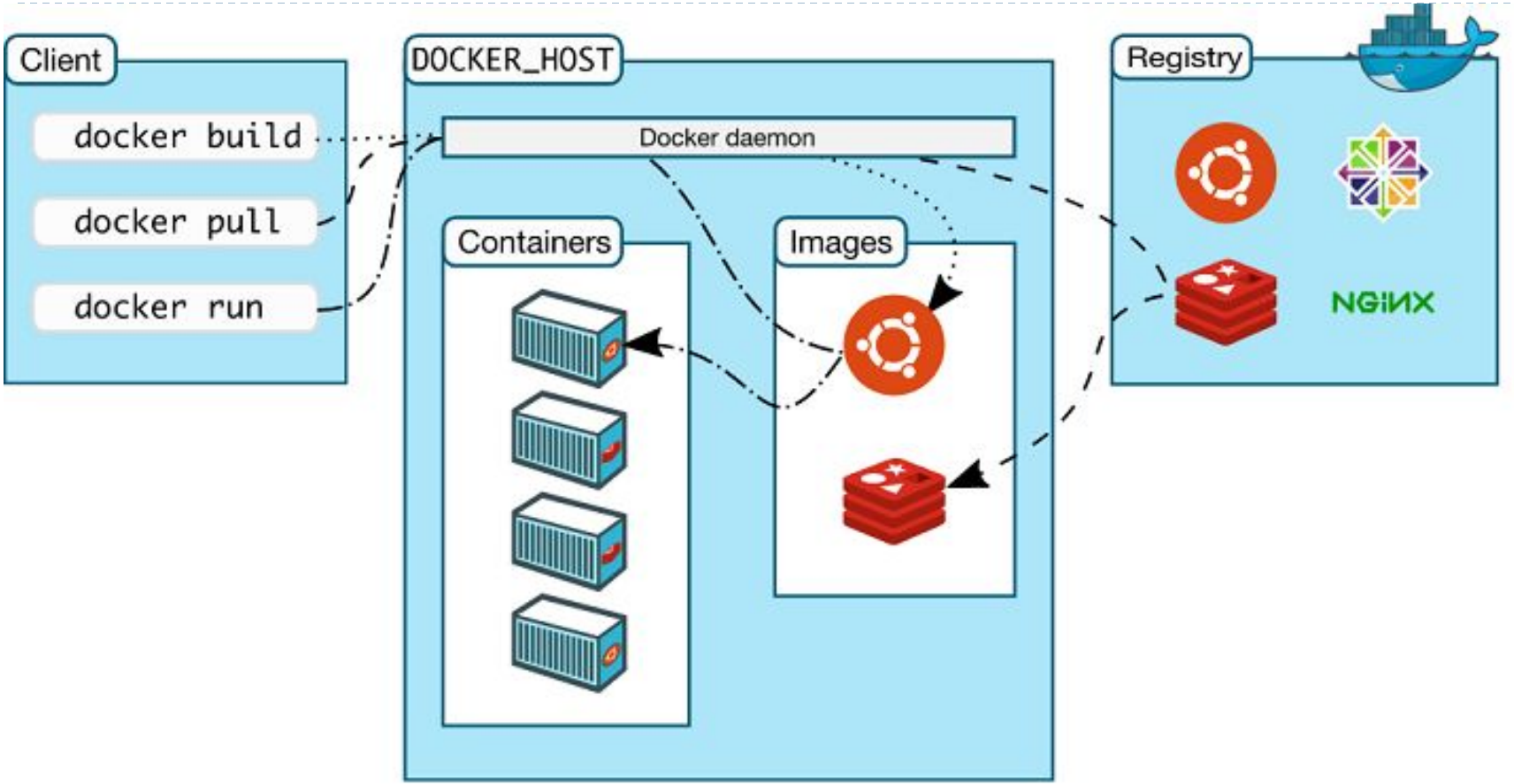
- Docker Container Lifecycle Management
- Portas
- Dockerfile



Opções

- **-d:** A opção -d (detached mode) faz com que o contêiner seja executado em segundo plano (ou seja, de forma "desanexada" do terminal).
- **-i:** A opção -i (interactive) mantém o STDIN aberto mesmo que o contêiner não esteja em anexo, permitindo a interação com o contêiner.
- **-t:** A opção -t (pseudo-TTY) aloca um terminal pseudo-TTY, que permite a interação com o terminal do contêiner.
- **-p:** Publica uma porta do contêiner no host. Por exemplo, -p 8080:80 mapeia a porta 80 do contêiner para a porta 8080 no host.
- **-v:** Monta um volume. Permite mapear um diretório do host para um diretório dentro do contêiner. Por exemplo, -v /meu/diretorio:/diretorio/no/conteiner.
- **--rm:** Remove o contêiner automaticamente quando ele for parado.
- **-e:** Define variáveis de ambiente dentro do contêiner. Por exemplo, -e MINHA_VARIAVEL=valor.
- **--network:** Conecta o contêiner a uma rede específica. Por exemplo, --network minha_rede.
- **--hostname:** Define o hostname do contêiner.
- **--restart:** Define a política de reinício para o contêiner (e.g., always, on-failure).
- **-u:** Define o usuário (UID ou nome) que o contêiner usará. Por exemplo, -u usuario.
- **--entrypoint:** Sobrescreve o ponto de entrada padrão da imagem.
- **--cpus:** Limita o número de CPUs que o contêiner pode usar. Por exemplo, --cpus="1.5" limita a 1.5 CPUs.
- **--memory:** Limita a quantidade de memória que o contêiner pode usar. Por exemplo, --memory="512m" limita a 512 MB de memória.
- **--link:** Conecta o contêiner a outro contêiner. Por exemplo, --link outro_conteiner.
- **-w:** Define o diretório de trabalho dentro do contêiner. Por exemplo, -w /meu/diretorio.
- **--name:** Define o nome do contêiner. Por exemplo, --name meu_conteiner.
- **--log-driver:** Define o driver de log a ser usado para o contêiner. Por exemplo, --log-driver=syslog.

Docker Arquitetura

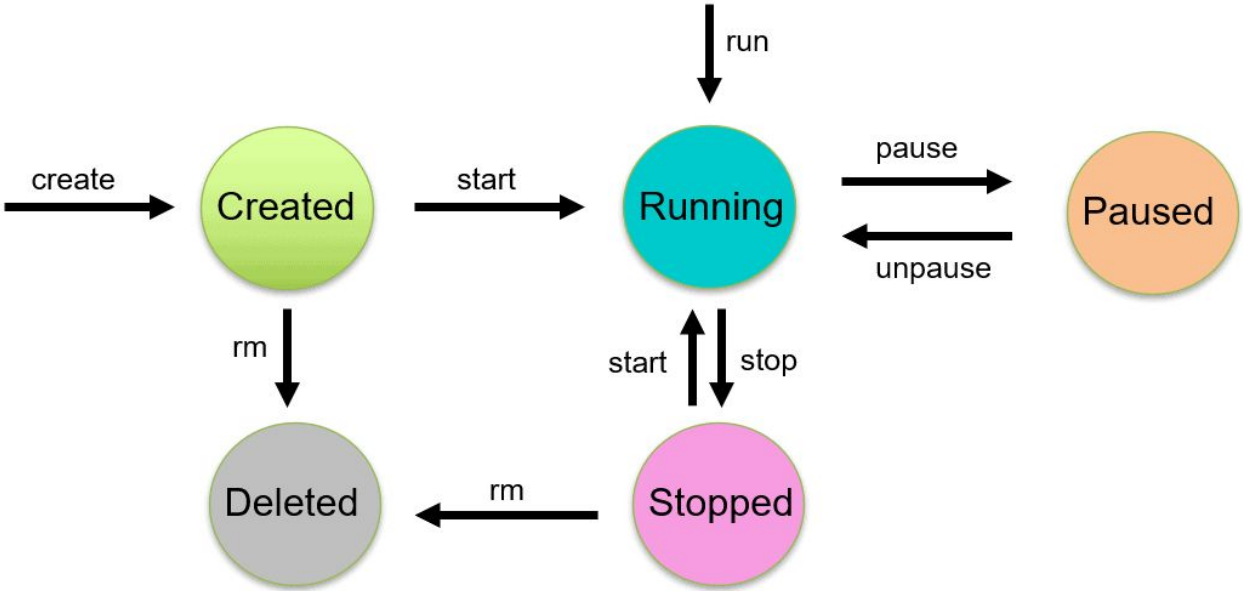


Docker Container Lifecycle Management

- **Docker Container Lifecycle Management** refere-se ao ciclo de vida de um contêiner no Docker, que abrange todas as etapas desde a criação até a exclusão de um contêiner.

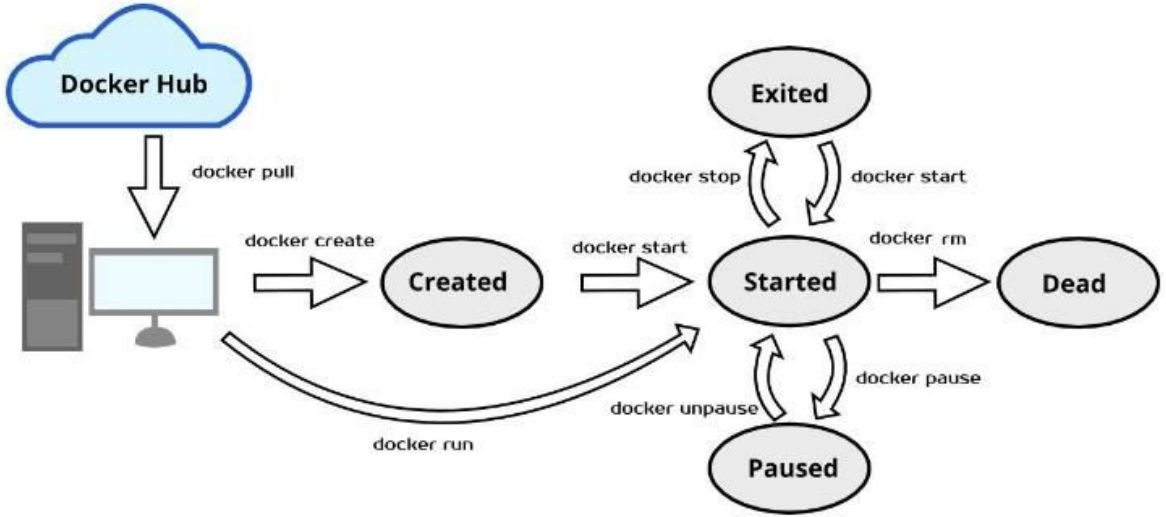
- Esse ciclo de vida é gerenciado por meio de comandos específicos que permitem **iniciar, parar, pausar e remover** contêineres conforme necessário. Vamos entender as etapas principais do ciclo de vida de um contêiner Docker.

Docker Container Lifecycle Management



- **Created:** Um contêiner que foi criado, mas não iniciado.
- **Running:** Um contêiner em execução com todos os seus processos ativos.
- **Paused:** Um contêiner cujos processos foram pausados.
- **Stopped:** Um contêiner cujos processos foram interrompidos.
- **Deleted:** Um contêiner em estado inativo (morto).

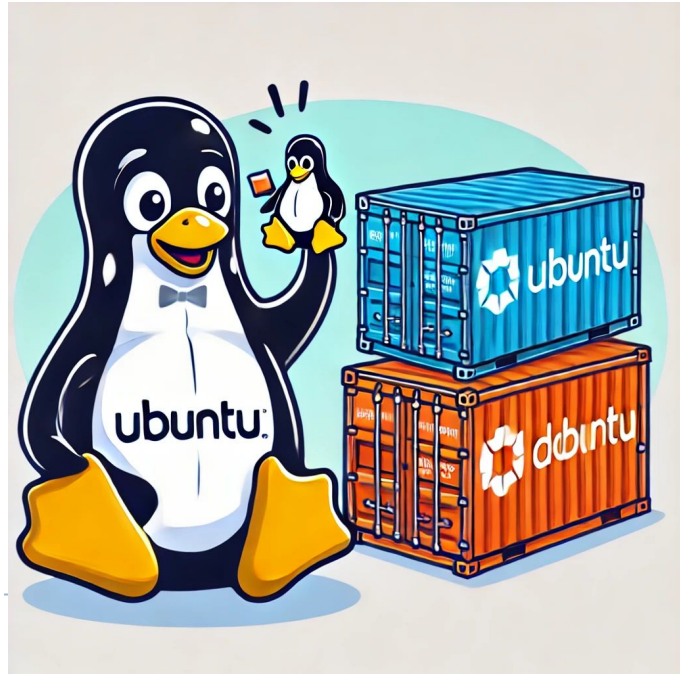
Docker Container Lifecycle Management



- **Created:** Um contêiner que foi criado, mas não iniciado.
- **Running:** Um contêiner em execução com todos os seus processos ativos.
- **Paused:** Um contêiner cujos processos foram pausados.
- **Stopped:** Um contêiner cujos processos foram interrompidos.
- **Deleted:** Um contêiner em estado inativo (morto).

Docker Container Lifecycle Management

- $ docker create --name <container name> <image name>
- $ docker start <container name>
- $ docker run -it --name <container name> <image name>
- $ docker pause <container name>
- $ docker unpause <container name>
- $ docker stop <container name>
- $ docker stop $(docker container ls –aq)
- $ docker rm <container name>
- $ docker ps -a
- $ docker rm $(docker ps -aq)
- $ docker kill <container name>

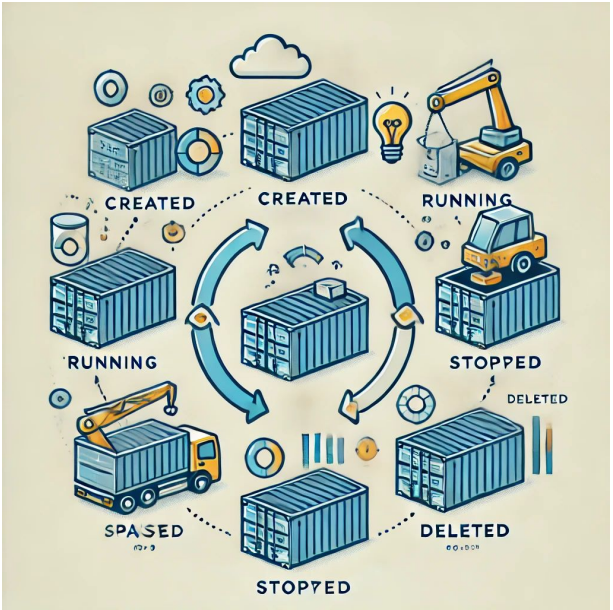


Docker Container Lifecycle Management

- **Docker rm vs. Docker Kill**

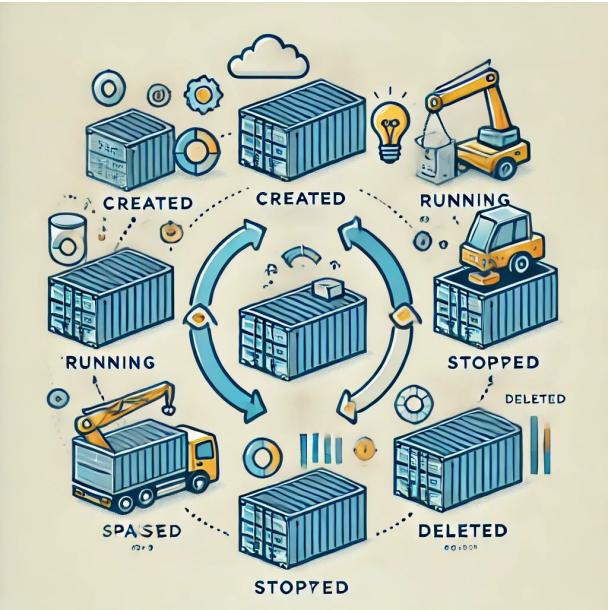
- **docker container rm:**
  - Usando docker rm, podemos remover um ou mais contêineres do nó host, e para isso pode-se usar o nome ou o ID do contêiner.

- **docker container kill:**
  - O processo principal dentro de cada contêiner especificado será enviado um sinal SIGKILL ou qualquer outro sinal especificado com a opção --signal.

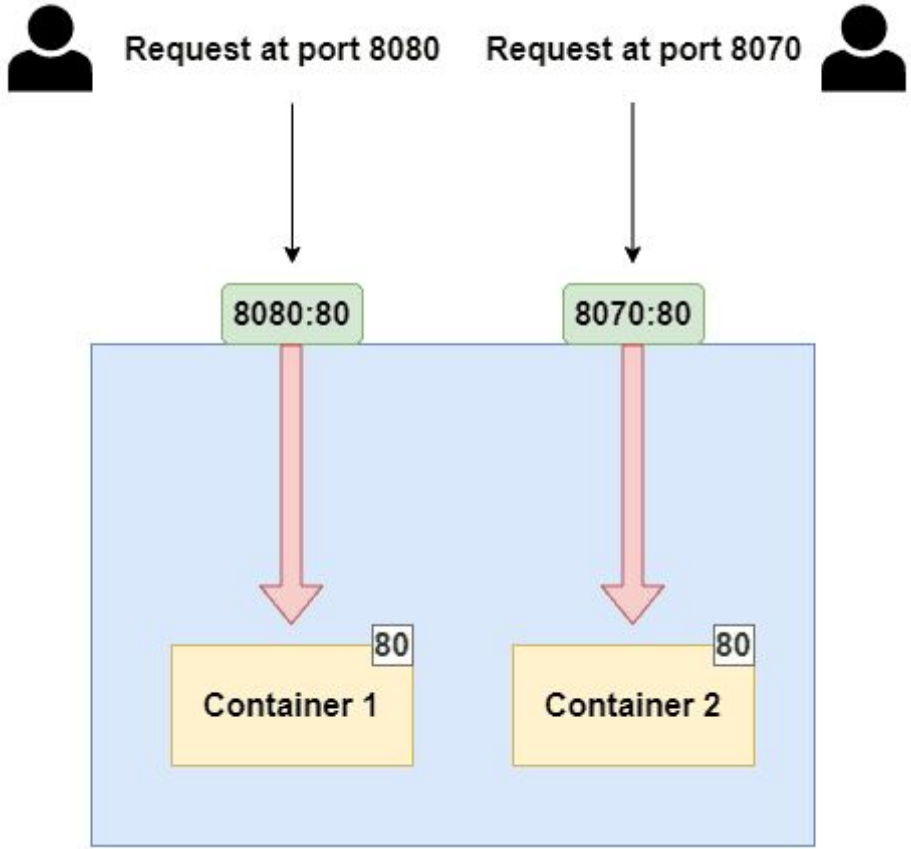


Docker Container Lifecycle Management

- No Docker, um contêiner é um **ambiente isolado** que executa aplicativos e serviços, semelhante a uma máquina física ou virtual.
- Assim como essas máquinas, os contêineres **têm seu próprio conjunto de portas.** No entanto, por padrão, essas portas não são diretamente acessíveis de fora do contêiner.
- É aqui que o mapeamento de portas, também conhecido como redirecionamento de portas, entra em ação.
- O mapeamento de portas permite que você exponha as portas dentro de um contêiner Docker, tornando os serviços em execução dentro do contêiner acessíveis ao sistema host ou a outros contêineres dentro do ambiente Docker.

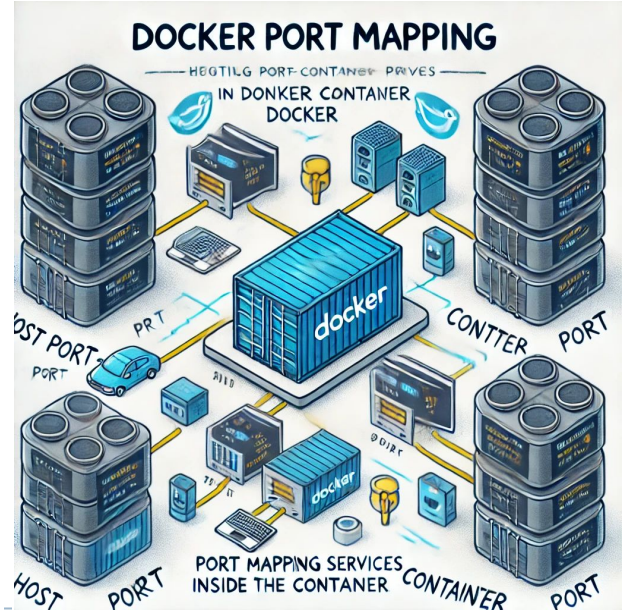


Docker Container Lifecycle Management



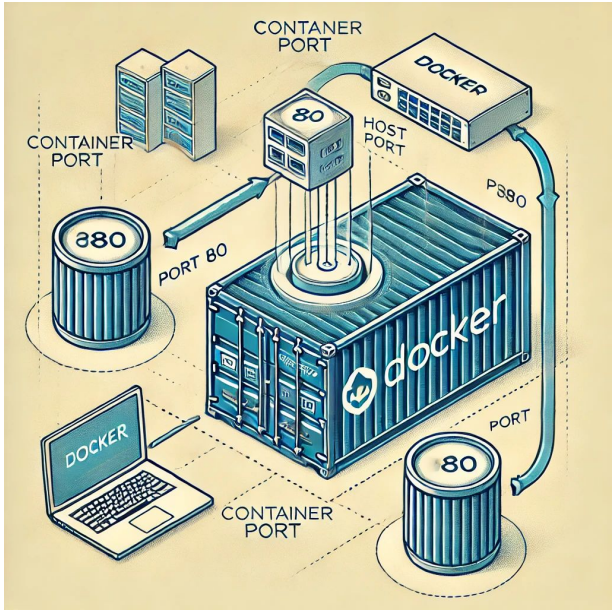
Key Concepts in Port Mapping

- **Conceitos-chave no Mapeamento de Portas:**
- **Porta:**
  - Portas são identificadores numéricos usados para diferenciar entre diferentes serviços de rede em execução no mesmo host. As portas são categorizadas em dois grupos:
    - **portas conhecidas** (que variam de **0 a 1023**) e **portas dinâmicas ou privadas** (que variam de **1024 a 49151**).
    - **As portas conhecidas são reservadas** para serviços comumente usados, como HTTP (porta 80) e HTTPS (porta 443).



Key Concepts in Port Mapping

- **Porta do Host vs. Porta do Contêiner:**
- **Porta do Host:**
  - Esta é uma porta no sistema host, que é a máquina onde o Docker está sendo executado. A porta do host é usada para acessar o serviço ou aplicativo em execução dentro do contêiner Docker a partir da máquina host ou de sistemas externos.
- **Porta do Contêiner:**
  - Esta é a porta na qual o serviço ou aplicativo dentro do contêiner Docker está ouvindo. Pode ser uma porta específica exigida pelo aplicativo, por exemplo, um servidor web ouvindo na porta 80.



- **Mapeamento Dinâmico:** Se você omitir a host_port, o Docker atribuirá automaticamente uma porta disponível no host. Isso é útil quando você deseja evitar conflitos de portas no sistema host.

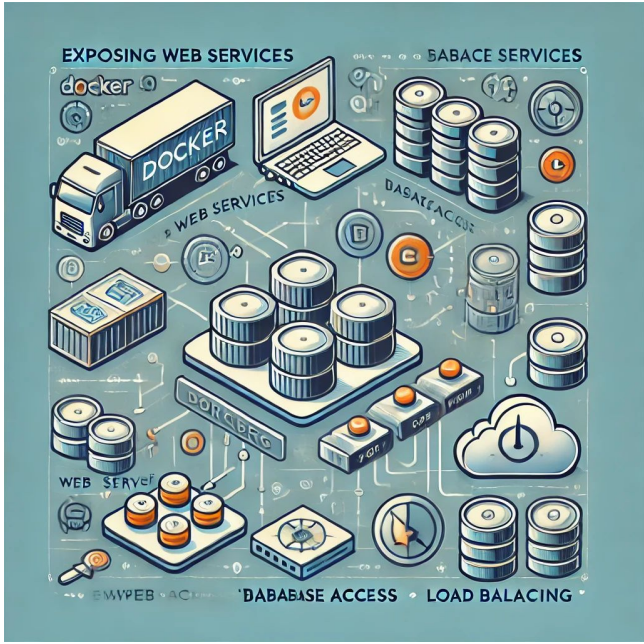
Mapeamento de Portas

- **Casos de Uso Comuns para Port Mapping (Mapeamento de Portas):**

- **Exposição de Serviços Web:** O mapeamento de portas é frequentemente utilizado para expor serviços web que estão sendo executados dentro de contêineres. Isso inclui servidores web como Nginx ou Apache, ou aplicativos web. O mapeamento de portas permite que os usuários acessem esses serviços através de um navegador web.

- **Acesso a Banco de Dados:** O mapeamento de portas pode ser usado para conectar contêineres a servidores de banco de dados. Isso permite que os aplicativos executados em contêineres interajam com serviços de banco de dados que estão fora dos contêineres.

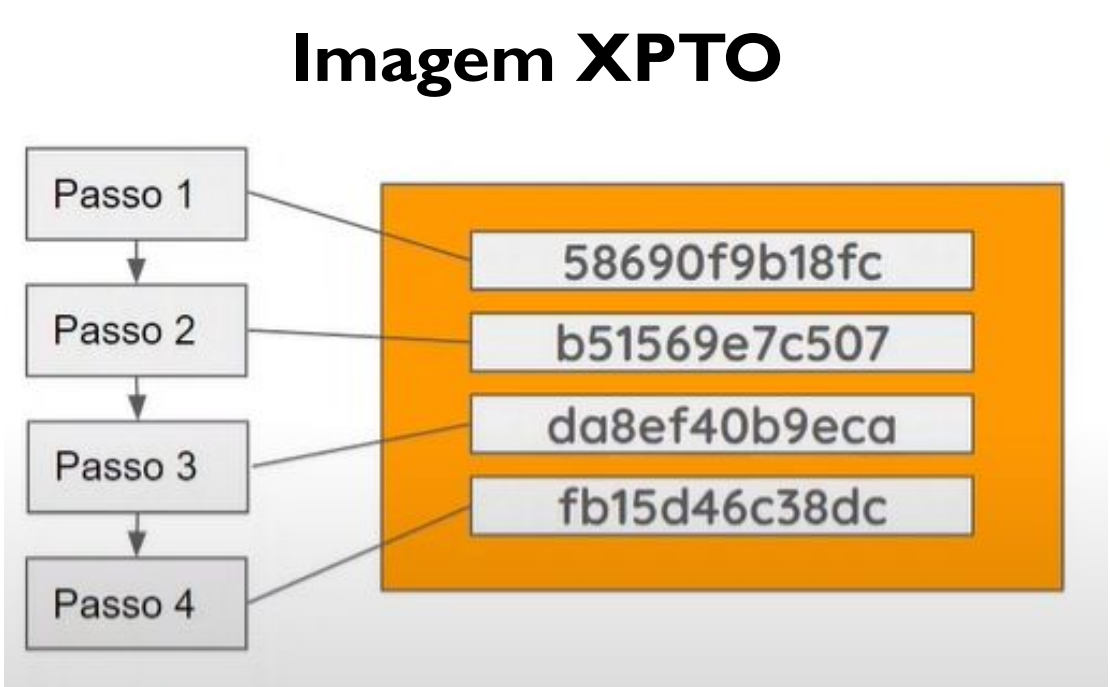
- **Balanceamento de Carga:** O mapeamento de portas desempenha um papel nas estratégias de balanceamento de carga, onde vários contêineres que fornecem o mesmo serviço são mapeados para a mesma porta no host. Isso garante a distribuição equilibrada das requisições recebidas.



Relembrando

- **Imagens são como receita de bolo**
  - seguir os passos para a montagem do bolo

Relembrando



- **Sequência de passos**
  - Comportamento de um contêiner
  - **4 passos**
    - São as camadas da imagem

Relembrando

- **Criação de um container**
  - docker run nome-da-imagem

- **Como ver as camadas de uma imagem do Docker hub?**
  - docker images
  - docker history dockersamples/static-site

Como criar uma imagem docker?

- **Como criar uma imagem docker?**
  - imagem base X
    - Cria a imagem Y
- **Tem como criar uma imagem do zero?**
  - Debian
  - Alpine
  - outros…

Entendo o docker commit

- **Aviso**
  - **Muito custoso**
  - **Não é tão prático de se trabalhar**
  - **É importante saber da sua existência e do seu funcionamento**

- **Dockerfile**

Docker commit:

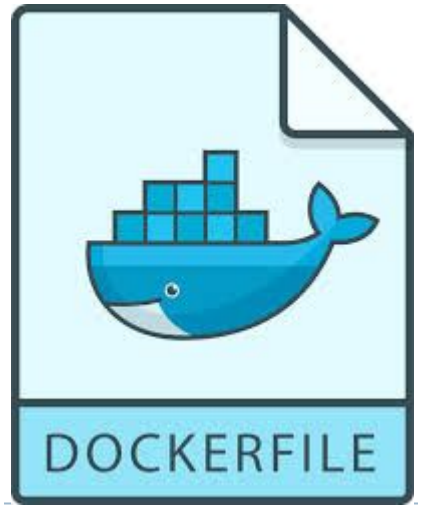
- docker run -it ubuntu
  - apt update
  - apt install curl ping vim
  - exit
- docker commit nome-container nome-nova-imagem
- docker images (vizualizar que as imagens estão com tamanhos diferentes)
- docker run -it --name nome-container nome-nova-imagem
- docker history nome-nova-imagem
- **Desvantagens do Docker Commit no modo interativo**
  - Camada muito pesada
  - Ações realizadas totalmente implícitas
- **Solução:**
  - **A cada alteração, realizar um commit**

Docker commit:

- Ubuntu
  - Entrypoint: **/bin/sh -c**
  - Comando padrão: **/bin/bash**
- docker run -it ubuntu ls
- docker history nova-imagem (camada muito grande e implícita não mostra os pacotes instalados)

Dockerfile

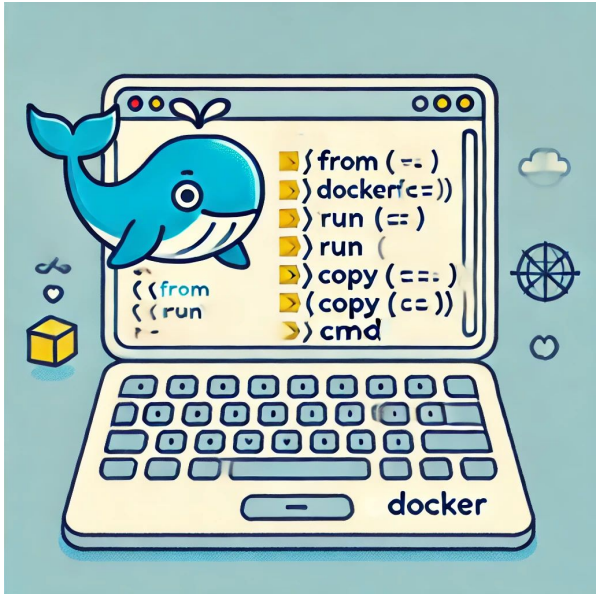
- Um Dockerfile é um arquivo de texto que contém todas as instruções necessárias para montar uma imagem Docker. Essas instruções definem a base da imagem, quais pacotes instalar, variáveis de ambiente, portas expostas, etc.



Dockerfile

- Quando você executa o comando docker run, o Docker utiliza a imagem já criada para iniciar um contêiner. No entanto, para criar a imagem, o Docker usa um arquivo chamado Dockerfile.

- O Dockerfile contém uma série de instruções que definem como a imagem deve ser construída, incluindo a instalação de pacotes, configuração do sistema e outras dependências necessárias para a aplicação.

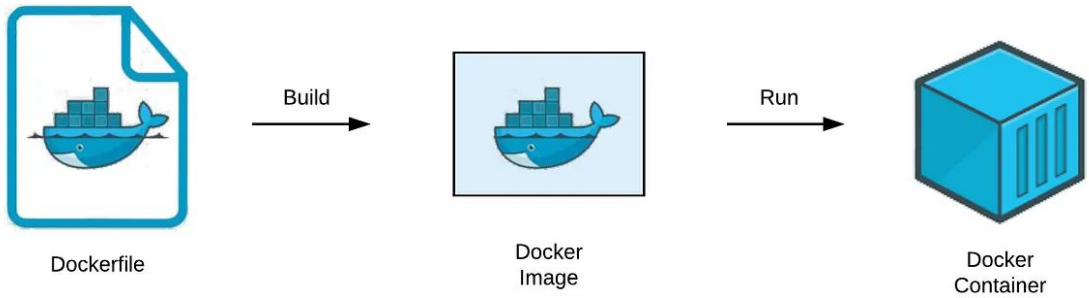


Benefícios de usar um Dockerfile

- **Automatização:** O Dockerfile permite automatizar o processo de construção da imagem. Toda vez que o Dockerfile é executado, ele cria a imagem do zero, garantindo que ela esteja atualizada com as versões mais recentes dos pacotes e dependências.
- **Segurança:** Como o Dockerfile compila a imagem a partir de zero cada vez que é executado, ele garante que você sempre tenha as versões mais atualizadas e seguras dos aplicativos. Isso minimiza o risco de incluir software desatualizado ou vulnerável na imagem.
- **Consistência:** O uso de um Dockerfile garante que a mesma imagem pode ser recriada em diferentes ambientes de desenvolvimento, teste e produção, mantendo consistência e previsibilidade.
- **Facilidade de Atualização:** Ao modificar o Dockerfile, você pode rapidamente reconstruir a imagem com as novas alterações, facilitando a atualização de aplicações e dependências.
- **Bem mais prático, controle de versão, colaboração, legibilidade**

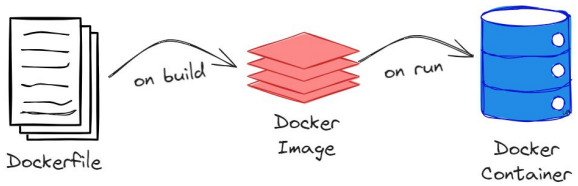
Dockerfile

- Criar os nomes dos arquivos de forma padrão.
- Dockerfile
- ou
- banco.dockerfile
- app.dockerfile
- api.dockerfile



Dockerfile

```
FROM ubuntu:22.04
RUN apt-get update && \
    apt-get install python3.11 python3.11-dev python3-pip -y
WORKDIR /app
COPY . .
RUN pip3 install --no-cache-dir -r requirements.txt
EXPOSE 8080
ENV LOGOMARCA="iamgem(link)"
ENV FOTO="iamgem(link)"
ENV NOME="GUTO"
ENV IDADE="GUTO"
ENV EMAIL="GUTO"
ENV PROFISSAO="GUTO"
ENV SITE="GUTO"
CMD [ "python3", "app.py" ]
```



Comandos

python3 -m venv venv

sudo apt install python3.10-venv

source venv/bin/activate

Instalar dependências no python

pip3 install --no-cache-dir -r requirements.txt

pip install --upgrade flask

python app.py

pip freeze > requerements.txt

pip show werkzeug

deactivate

docker build -t minha-imagem:v1 . (o comando tem o ponto (.) de pasta local)

docker images

docker run -it -p 8080:8080 minha-imagem:v1

MUITO OBRIGADO!!!!!

Guto Muniz

augustomuniz@gmail.com